DAT405/Assignment 8 - Rule-based AI

Francisco Boudagh
Jakob Engström
Group 2

May, 2023

Question 1

Through testing with various directed graphs, we have observed that the number of iterations increases exponentially with the length r. For each layer, the number of iterations becomes d^r, where r represents the length to the goal. To determine the total number of max iterations, denoted as I_{max}, we simply need to sum up the iterations for each layer, as shown in equation 1.

$$I_{max} = \sum_{i=1}^{r} d^i \tag{1}$$

This is a geometric sum and can be written as:

$$I_{max} = \frac{d^{r+1} - d}{d - 1}, \qquad d \neq 1 \tag{2}$$

For instance, consider a perfect directed graph with four layers as shown in figure 1 below. If we wish to travel from node 1 to node 15, the number of iterations would be 14, which can be easily counted.

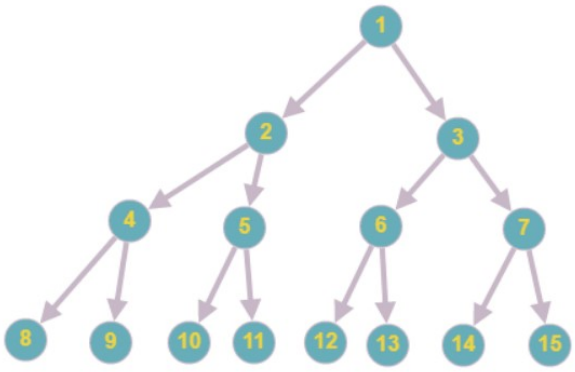


Figure 1: Perfect directed graph with a maximum of 2 children and 4 layers in total.

Now, let's test our formula to verify this. In this particular case, $d = 2$ and $r = 3$.

$$I = \frac{d^{r+1} - d}{d - 1} = \frac{2^{3+1} - 2}{2 - 1} = \frac{2^4 - 2}{1} = 14. \tag{3}$$

b) Memory required for BFS

Since the number of nodes is equal to the number of iterations plus 1 for the first node, we can represent the number of nodes as $k = I+1$. Considering that each node stores 1 unit of memory, we can calculate the maximum amount of memory required as follows:

$$M = I + 1 = \frac{d^{r+1} - d}{d-1} + 1 = \frac{d^{r+1} - 1}{d-1}. \tag{4}$$

Question 2

Let us call the three unknow nodes X, Y and Z as shown in figure 2. Depth First Search (DFS) algorithm stops searching and visiting nodes when all the nodes are visited. DFS can get stuck in ties, for example in the graph in figure 2 between X-Y-Z. In our task, we have implemented a tiebreaker rule for DFS, which is choosing label with the smallest value. Without this rule the DFS can get stuck in never-ending-loops. Now we have to label X-Y-Z so DFS never get stuck.

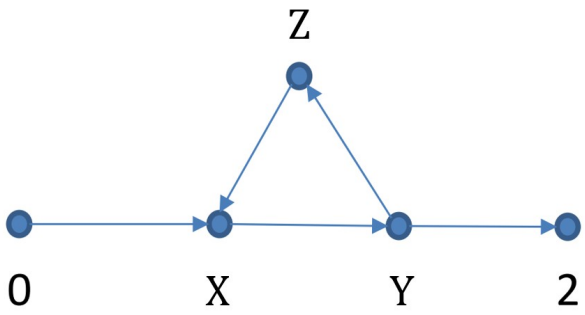


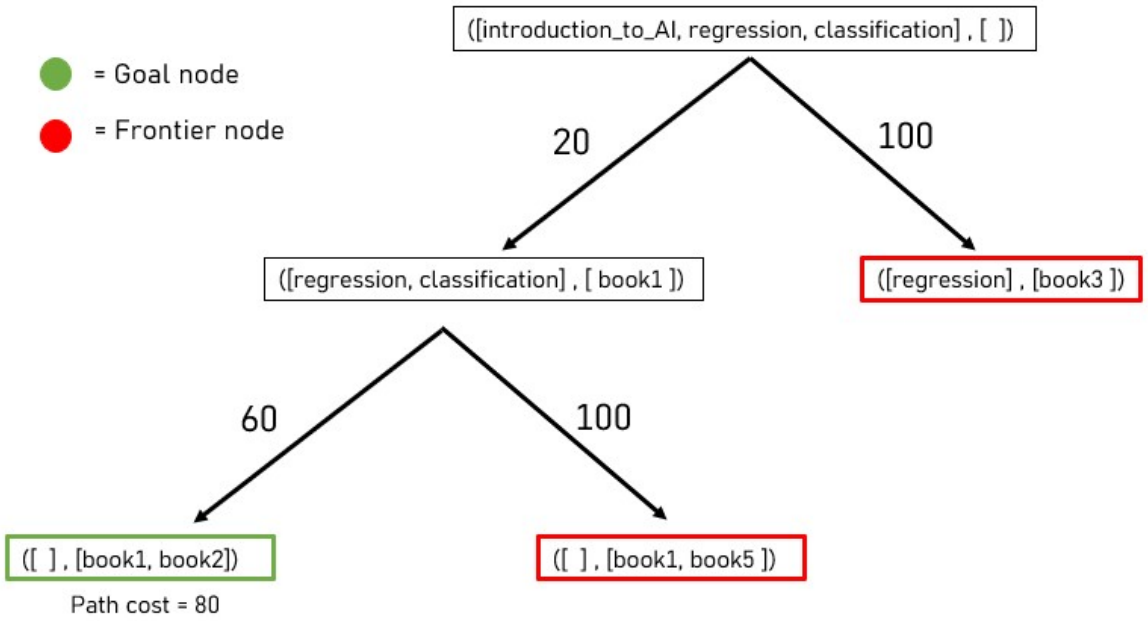
Figure 2: Graph with two known nodes (intial 0 and goal 2) and three unknown nodes (X,Y,Z).

The nodes X, Y, and Z have the labels 1, 3, and 4, respectively, but it is unknown which node corresponds to each value. After traveling from 0 to X and from X to Y, as the arrows shows us, then from Y we have a choice to move to Z or 2. So, Z should have a value less than 2 to avoid loops. With our labels (1, 3 and 4) this is means choosing Z not equal to 1 (leading Z being 3 or 4) ensures that the DFS never gets stuck. Below we have all the different combinations.

(X, Y, Z) = (1, 3, 4) → OK (Z!=1)
(X, Y, Z) = (1, 4, 3) → OK (Z!=1)
(X, Y, Z) = (3, 1, 4) → OK (Z!=1)
(X, Y, Z) = (3, 4, 1) → Stuck (Z=1)
(X, Y, Z) = (4, 1, 3) → OK (Z!=1)
(X, Y, Z) = (4, 3, 1) → Stuck (Z=1)

Question 3

a) Search space tree



By evaluating from the leftmost topic in each node, choosing a book and following the least cost path we end up finding the Goal node at a cost of 80. The other unexplored nodes becomes frontier nodes. The node directly to the right of the Goal node is also a Goal node, but since we followed the lowest cost path, we haven't "explored" this node yet, hence it is a frontier node.

b) Non-trivial heuristic function

Admissibble heurisitic functions work by exploring the node that is nearest to the goal. This could be achieved by looking at the amount of topics left in the node, choosing to progress with the one having the fewst topics left, since the goal is to empty the topics bracket.

Question 4

The Manhattan distance in this question will be denoted as MD.

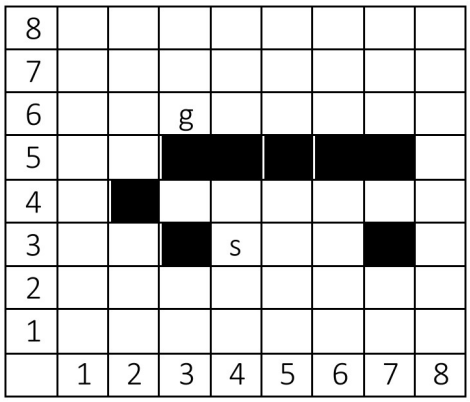


Figure 3: 8x8 grid with start and goal marked as s and g. Black boxes are walls.

a) First five iterations of A*

Iteration 1:
start - (43)
and chooses (43)
Path: {43}

Iteration 2:
(43/s) explores:
(42) MD = 5
(44) MD = 3
(53) MD = 5
and chooses (44)
Path: {43, 44}

Iteration 3:
(44) explores:
(43/s) MD = 4
(34) MD = 2
(54) MD = 4
and chooses (34)
Path: {43, 44, 34}

Iteration 4:
34 can only explore already explored so it goes back to 44 which is also explored, goes further back to (43/s).
(43/s) explores:
(42) MD = 5
(44) MD = 3, already visited
(53) MD = 5
and chooses (42)
Path: {43, 42}

Iteration 5:
(42) explores:

(41) MD = 6
(32) MD = 4
(43/s) MD = 4
(52) MD = 6
and chooses (32)
Path: {43, 42, 32}

b) Github A* vs Breadth-First-Search vs Best-First-Search

No diagonals, only top-right-down-left. Manhattan distance choosed.

A*
Starting like our first five iterations in a).
Length: 10
Operations: 71

Breadth-First-Search
Length: 10
Operations: 364

Best-First-Search
Length: 10
Operations: 48

All three algorithms find the best path with a length of 10. Breadth-First-Search performs the worst, with most operations, 364. Best-First-Search performs the fewest operations, only 48, making it the best performing algorithm in this case.

Breadth-First Search doesn't consider the distance to the goal, it simply explores all explorable cells until it reaches the goal, which explains why it have more operations.

The main difference between A* and Best-First-Search is that A* considers both the remaining distance to the goal $h(n)$ and the cost of reaching the current node from the starting point $g(n)$. By combining these two factors using the evaluation function $f(n) = g(n) + h(n)$, A* can often explore better paths towards the goal.

In summary, Breadth-First-Search explores all nodes without using any heuristic information and lacks knowledge about the goal. Best-First-Search uses a heuristic function that estimates the distance from a node to the goal $h(n)$. A* combines Best-First-Search with information about the distance traveled so far $g(n)$, making it more effective at finding the optimal path.

c) Breadth-First-Search sometimes better than Best-Frist-Search

Best-First-Search can create a longer path to the goal than Breadth-First-Search if there are walls in the direction closer to the goal. For example, that Best-First-Search thinks it is getting closer to the goal, because the distance to the goal is decreasing all the time, but then all of a sudden there is a wall that it has to go around. In the example shown in figure 4, Breadth-First-Search's path (blue) is 22 long while Best-First-Search (orange) find a 28 long path. However, Best-First-Search is much faster since it explores much less cells than Breadth-First-Search.

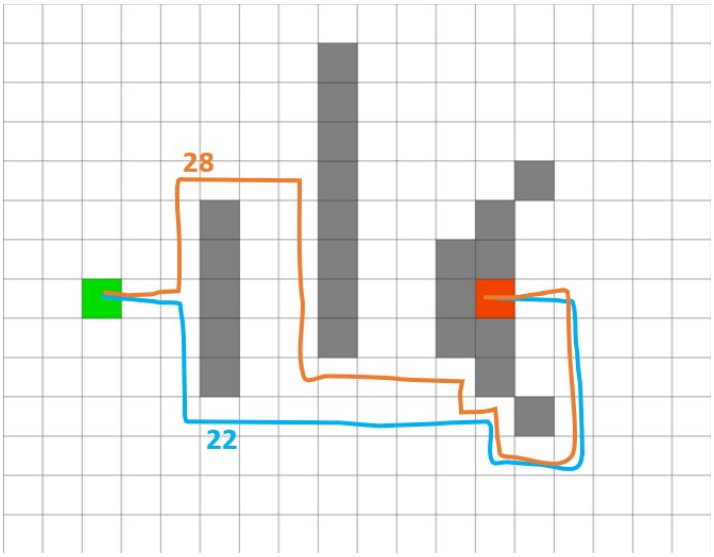


Figure 4: Grid with walls. Orange: Best-First-Search. Blue: Breadth-First-Search.